# Detailed Explanation of Diffusion Process and Corresponding Code

# 1 Method

We start by summarizing the diffusion models in section 1.1.1 and guidance to control the generation process in section 1.1.2. In **??** we describe our framework to incorporate control from two unpaired datasets.

## 1.1 Background

### 1.1.1 Diffusion models

Diffusion Models (DMs) [**?**] are generative models that generate data samples by gradually adding noise to data through a forward diffusion process, followed by a reverse denoising process that reconstructs the original sample. The forward process corrupts data sample $x_0$ through iterative noise addition controlled by a schedule $\alpha = \{\alpha_t\}_{t=1}^{T}$:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\alpha_t}\, x_{t-1}, \alpha_t I) \tag{1}$$

With $\bar{\alpha}_t = \prod_{i=1}^{t}(1-\alpha_i)$ noised $x_t$ can be computed from $x_0$ with marginal distribution given by,

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t) I) \tag{2}$$

The reverse process, parameterized by a neural network $p_\theta$, learns to reconstruct the data sample by predicting the denoised mean and variance at each step:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{3}$$

Latent Diffusion Models (LDMs) work in a compressed latent space $z_t$ rather than the high-dimensional data space, improving efficiency[**?**]. The data $x_0$ is encoded as $z_0$ through an autoencoder, and the diffusion process is then applied in this latent representation. LDM models learn to minimize the objective,

$$L = \mathbb{E}_{z_0,t,\epsilon\sim\mathcal{N}(0,1)}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|_2^2\right] \tag{4}$$

where, $\epsilon$ is random Gaussian noise sampled from $\mathcal{N}(0, I)$, and $\epsilon_\theta$ is the model's predicted noise at time $t$. Owing to the relevance to score-based generation models, the model estimates the log density of the distribution $z_t$ i.e. $\epsilon_\theta(z_t, t) \approx -\nabla_{z_t} \log p(z_t)$.

### 1.1.2 Guidance

In the denoising process different conditional inputs $c$ (text, image, depth, mask, etc.) can be added to control the generation; so the denoising model predicts $\epsilon_\theta(z_t, t, c)$. At the time of sampling, diffusion score $\epsilon_\theta(z_t, t, c)$ is modified to include the adversarial gradient of the classifier[**?**]:

$$\tilde{\epsilon}_\theta(z_t, t, c) = \epsilon_\theta(z_t, t) - w\nabla_{z_t} \log p_\phi(c|z_t) \tag{5}$$

where $w$ is a guidance strength parameter that controls the influence of the classifier. In classifier-free guidance [**?**] single neural network is used to parameterize both the unconditional denoising diffusion model $p_\theta(z)$ and conditional denoising diffusion model $p_\theta(z|c)$. While training unconditional model receives a null token, $\Phi$ as $c$ randomly with some probability $p_uncond$, set as a hyperparameter. While sampling linear combination of conditional and unconditional score estimates is used:

$$\tilde{\epsilon}_\theta(z_t, t, c) = (1 + w)\epsilon_\theta(z_t, t, c) - w\epsilon_\theta(z_t, t) \tag{6}$$

# 2 Forward Diffusion (Noising Process)

## 2.1 Step 1: Noise Addition

The forward diffusion process incrementally adds Gaussian noise to the data:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\alpha_t} \cdot x_{t-1}, \alpha_t \cdot I)$$

## 2.2 Direct Sampling of $x_t$

The noise is accumulated across timesteps, allowing $x_t$ to be sampled directly from $x_0$:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} \cdot x_0, (1-\bar{\alpha}_t) \cdot I)$$

## 2.3 Code Implementation:

```
alphas = 1.0 - betas
self.alphas_cumprod = np.cumprod(alphas, axis=0)  # \(\bar{\alpha}_t\)
self.sqrt_alphas_cumprod = np.sqrt(self.alphas_cumprod)  # \(\sqrt{\bar
    {\alpha}_t}\)
self.sqrt_one_minus_alphas_cumprod = np.sqrt(1.0 - self.alphas_cumprod)
     # \(\sqrt{1 - \bar{\alpha}_t}\)

def q_sample(self, x_start, t, noise=None):
    if noise is None:
        noise = th.randn_like(x_start)  # \(\epsilon \sim \mathcal{N
            }(0, I)\)
    return (
        _extract_into_tensor(self.sqrt_alphas_cumprod, t, x_start.shape
            ) * x_start
        + _extract_into_tensor(self.sqrt_one_minus_alphas_cumprod, t,
            x_start.shape) * noise
    )
```

# 3 Reverse Diffusion (Denoising Process)

## 3.1 Step 1: Reverse Distribution

The reverse process learns:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$$

## 3.2 Code for Sampling $x_{t-1}$:

```
def p_sample(self, model, x, t, clip_denoised=True, denoised_fn=None,
    cond_fn=None, model_kwargs=None):
    out = self.p_mean_variance(
        model,
        x,
        t,
        clip_denoised=clip_denoised,
        denoised_fn=denoised_fn,
        model_kwargs=model_kwargs,
    )
    noise = th.randn_like(x)
    nonzero_mask = (t != 0).float().view(-1, *([1] * (len(x.shape) - 1)
        ))  # no noise when t == 0
    sample = out["mean"] + nonzero_mask * th.exp(0.5 * out["
        log_variance"]) * noise
    return {"sample": sample, "pred_xstart": out["pred_xstart"]}
```

—

## 3.3 Step 2: Predicted Noise $\epsilon_\theta(x_t, t)$

The model directly predicts the noise added to $x_t$:

$$\epsilon_\theta(x_t, t) = f_\theta(x_t, t)$$

## 3.4 Code:

```
model_output = model(x_t, self._scale_timesteps(t), **model_kwargs)  #
    \(\epsilon_\theta(x_t, t)\)
```

—

## 3.5 Step 3: Predicted $x_0$

Using the predicted noise $\epsilon_\theta(x_t, t)$, the noiseless image $x_{\text{pred}_0}$ is reconstructed:

$$x_{\text{pred}_0} = \frac{x_t - \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}$$

### 3.6 Code:

```
def _predict_xstart_from_eps(self, x_t, t, eps):
    return (
        _extract_into_tensor(self.sqrt_recip_alphas_cumprod, t, x_t.
            shape) * x_t
        - _extract_into_tensor(self.sqrt_recipm1_alphas_cumprod, t, x_t
            .shape) * eps
    )
```

---

### 3.7 Step 4: Predicted Mean $\mu_\theta(x_t, t)$

The mean of the reverse process is computed as:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \cdot \left( x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \cdot x_{\text{pred}_0} \right)$$

### 3.8 Code:

```
def q_posterior_mean_variance(self, x_start, x_t, t):
    posterior_mean = (
        _extract_into_tensor(self.posterior_mean_coef1, t, x_t.shape) *
            x_start
        + _extract_into_tensor(self.posterior_mean_coef2, t, x_t.shape)
            * x_t
    )
    posterior_variance = _extract_into_tensor(self.posterior_variance,
        t, x_t.shape)
    return posterior_mean, posterior_variance
```

---

### 3.9 Step 5: Adding Stochastic Noise

Finally, noise is added to the reverse mean to sample $x_{t-1}$:

$$x_{t-1} = \mu_\theta(x_t, t) + \sqrt{\Sigma_\theta(x_t, t)} \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

### 3.10 Code:

```
sample = out["mean"] + nonzero_mask * th.exp(0.5 * out["log_variance"])
    * noise
```

---

## 4 Summary Table of Code and Equations

| Mathematical Equation | Code Implementation |
|---|---|
| $q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t} \cdot x_0, (1 - \bar{\alpha}_t) \cdot I)$ | q_sample function. |
| $x_{\text{pred}_0} = \frac{x_t - \sqrt{1-\bar{\alpha}_t} \cdot \epsilon_\theta(x_t, t)}{\sqrt{\bar{\alpha}_t}}$ | _predict_xstart_from_eps. |
| $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \cdot \left( x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot x_{\text{pred}_0} \right)$ | q_posterior_mean_variance function. |
| $x_{t-1} = \mu_\theta(x_t, t) + \sqrt{\Sigma_\theta(x_t, t)} \cdot \epsilon$ | Final sampling step in p_sample. |